

# Intrusion Detection: Filling in the Gaps

Robert MacBride 2000-04-06

## Introduction

If you've recently purchased a new intrusion detection system, you're probably just beginning to realize how painful it is to take analysis beyond a window full of warning messages to meaningful information. And no matter how much marketing is done by companies to convince you otherwise, today's commercial products still have quite a way to go. After all, detecting events is only half the job; you still need to manage your systems, identify sources, and investigate the extent of attacks.

I'm one of those guys that never thinks the latest software has enough bells and whistles. My experience with intrusion detection software is no different. So rather than settle on what's available, I've developed some ways to help integrate heterogeneous products, and as a result, have found a new realm of underutilized functionality. Bear in mind that homemade integration is not for the programming squeamish of heart; it does take work.

These concepts are based on my own experiences with products on the market today and are ones that I myself use in practice. I've taken some effort not to mention any products by name, so as not to imply that you need to use the same products to get the same functionality. Don't assume, however, that every intrusion detection system can support all of these features. If you're still evaluating products, closely assess the flexibility of your top choices. You may actually find that some of the more commercialized products are too limiting.

## Things Even the Big Guys Forgot

First, let's discuss a few of the components missing from many of today's IDS systems. The most notable of which is the lack of standards between intrusion detection vendors. Just try to get a host-based IDS from one vendor to work with a network-based IDS from another company. Some promising initiatives are underway, such as the [Intrusion Detection Working Group \(IDWG\)](#) model for interoperability, but it may be quite a while still before we see a standard like this fully implemented industry-wide. This means that today, you cannot use one IDS interface to poll another vendor's real-time data, easily correlate results, or seamlessly integrate alert management out of the box.

Even the individual IDS systems are missing some basic functions. For instance, the easiest way to customize a product is to make event-driven calls to external programs. To do this right, it's really best to have some control over the parameters you pass to those programs. Good luck. None of the major players today provide a full set of user-retrievable variables (source/destination IP, usernames, flags, ports, etc), and some don't let you make calls to the command line at all.

If intrusion detection applications are to move to the next level, they will need to broaden their focus beyond mere signature matching to incorporation of long-term trend analysis, built-in investigative tools, and more robust administrative features. The ability to monitor the status of the IDS sensors themselves, automatically research source IP's, or create your own filters has been largely neglected. Keep some of these things in mind, and read on to find out how you can fill in some of the gaps in today's systems.

## A Few Words of Caution

The code snippets used here are just samples for you to try out; they are not meant to be fully developed programs. If you decide to write your own code to supplement features, avoid some common mistakes. Do a sanity check to verify that all input meets appropriate criteria before processing it. This will help prevent vulnerabilities that could allow others to execute code on your system. For instance, if you inadvertently pass a back tick ( ` ) into shell code, the system may interpret it as a request to execute the next thing it sees on the line.

Also, plan to build in "throttling"-the ability to limit how many alerts are sent in a given time period. If you don't have a means to prevent an unexpected flood of alerts, you could find yourself overflowing your email inbox or receiving thousand dollar pager bills!

## System Monitoring

When I first installed our shiny, new host-based detection system, I immediately realized that I would need some way to track the status (up or down) of agents deployed on a few hundred servers. Being a bit naïve, I asked tech support if there was a built-in tool to monitor the status of their remote sensor software. No such luck.

Most networked applications work on a simple principle-starting the remote program automatically opens a service port to listen for communications from the administrative console. Checking for the presence of this open port should give you a reasonable indication of the application's status. If the port does not respond, pinging the server or checking another port that is expected to be open can help distinguish between the whole system being down or just the application. While using this method does not guarantee 100% that the software is functioning properly, from my experience, it does provide a high degree of reliability in reporting communications issues. If your central console is on UNIX, you can use the snippet of code below to build your own port monitoring system. Caution: make sure that your network-based sensors are not reporting the connections as an anomaly; if you're using a dedicated LAN to segregate console communications and packet capturing, this shouldn't be a problem.

### Example 4-1

```
#!/bin/sh
# -----
# PORT MONITOR
# -----
TIMEOUT=5           # User-definable timeout (in sec) if no connection
PORT=22             # Port to monitor
HOST_LIST="fred barney 10.10.10.10"

# .....
connect_port() {    # Function to connect to a port
    # print a control-] to the telnet session to get command mode,
    # then quit the session.
    echo "\n^\nquit"| telnet ${HOST} ${PORT} > ${TMP_FILE} 2>/dev/null
```

```

}
# .....

for HOST in ${HOST_LIST}; do

    TMP_FILE="port_monitor.${HOST}.$$"
    cat /dev/null > ${TMP_FILE}          # Create a temp file to store results

    # Connect to the monitored port. Note that this is run as a background
    # process, so you don't have to wait for normal telnet timeout period.
    connect_port &

    COUNT=0
    while [ ${COUNT} -lt ${TIMEOUT} ]; do
        fgrep -i "connected" "${TMP_FILE}">/dev/null
        CONNECT_FOUND=${?}              # Check for a 'connect' message
        if [ ${CONNECT_FOUND} -eq 1 ]; then
            sleep 1
            COUNT=`expr ${COUNT} + 1`
        else
            break
        fi
    done

    echo "Status of port ${PORT} on host ${HOST}: \c"
    [ "${CONNECT_FOUND}" -eq 0 ] && echo "Up"   || echo "Down"
    rm -f "${TMP_FILE}"
done

```

## Adding Functionality to Host-based Systems

Most host-based systems are very good at monitoring files-permissions, checksums, etc-but they have little or no process monitoring capabilities. This can be supplemented fairly easy if you have an IDS that can be configured to monitor user-defined log files. Essentially, the idea is to periodically write a process list to file and allow the IDS to search for regular expressions. Pre-formatting the data properly will allow the IDS to handle the data easier. If you're scripting the process dump, you can also build the filtering directly into the code (see sample below). A variation on this code can also be used to monitor modem connections by inspecting the state of the getty process.

### Example 5-1

```
#!/bin/sh
# -----
# PROCESS MONITOR
# -----
DETECT_LIST="crack nmap rootkit" # Sample process detection list
IDS_LOG="processes.log"          # Log file monitored by IDS
TMP_FILE="ps.$$"                # Temporary file used to hold a copy
                                # of the process list. This is done
                                # to prevent multiple ps calls.
FOUND_MSG="Process Detected"    # Message that the IDS can monitor for if necessary

ps -ef > ${TMP_FILE}
for PROCESS in ${DETECT_LIST}; do
    FOUND_PS=`grep -i "${PROCESS}" ${TMP_FILE}` # Check for process
    [ -n "${FOUND_PS}" ] && echo "${FOUND_MSG}: ${FOUND_PS}" >> ${IDS_LOG}
done
rm "${TMP_FILE}"
```

## Creating Your Own Alert Logs

From this point on, I want to focus primarily on alert management and logging (i.e. handling all those messages that your IDS is creating). Why create your own alert logs? Until industry standards are set, this may be your best bet to integrate data from various sources. By doing so, you can sometimes enhance filtering capabilities and append information to alerts that your current IDS may not already provide. Examples of additional fields include hostname resolution or [Common Vulnerabilities and Exposures \(CVE\)](#) codes.

Before adding in your own customization, you need to get access to the data that your IDS gathers. It's best if your system can be configured to create the logs by calling an external log generation program. Otherwise, you will need to read from the original logs directly. Since we'll only be discussing alert logging here, the volume of data should remain manageable, so I suggest keeping things simple. Write out text files either in CSV (comma separated values) or tab delimited formats rather than dealing with binary data formats. This fundamental choice will allow easier database loading, spreadsheet imports, and conversion into HTML. Make sure you have each field format well defined for consistency and allow several free-form fields at the end for anything that doesn't fit into the predefined categories.

Below is a sample format:

Date,Time,Priority,IDS,Detection Source,Attack Type Code,Attack Name,CVE,Source Username,Destination Username,Source Hostname,Destination Hostname,Source IP,Destination IP,Source Port(s),Destination Port(s),Freeform fields?

### Example 6-1

```
1999-03-20, 14:20,6, ABC NID, dino, 3, PHF Attack, CVE-1999-0067,,,dialup13.hackem.net, Barney, 10.31.33.7,10.10.10.10, 42524,8080, /cgi-bin/phf%20cat%20/etc/passwd
```

### Example 6-2

```
1999-03-20, 14:20, 5 , XYZ HBID, fred dino, 2, Invalid su attempts,,wilma, root sys bin ftp, fred dino, fred dino, 10.10.10.10 10.10.10.20, 10.10.10.10 10.10.10.20,pty3 pty7,pty3 pty7,20 attempts,4 users,2 systems
```

In the first example (6-1), the dialup host at *hackem.net* (IP address 10.31.33.7) attempted to exploit a PHF vulnerability (listed with CVE #1990-0067) against the host Barney (IP address 10.10.10.10). The scan was detected by the network-based IDS system called ABC on the sensor named Dino that classified it as a "PHF Attack" and gave it a priority rating of 6. Note that we coded the attack type as 3, which in our scheme might translate to something like "known exploit". We could have also coded the attack name. Using translation codes instead of actual descriptions reduces the log size and allows you to use a mapping file to get consistent descriptions. The freeform fields contain additional information to specify what the user attempted to execute.

In the second example (6-2), the user *wilma* attempted to su to 4 different users for a total of 20 combined failed attempts on 2 systems (fred and dino). The event was picked up by XYZ host-based IDS which classified it as type 2 (translated as "unauthorized access attempt"). Note that the 'port' field refers to a terminal (tty) port rather than a service port. The freeform fields provide information related to invalid logins as: # of attempts, # of target users, # of target systems.

## Creating Your Own Alert Management System

Once you're able to generate alert logs in your own format, most of the hard work is done. You can now design an integrated alert management system. This system can be implemented in one of two modes—batch mode or real-time mode. To use batch mode, it's easiest to move the active (original) log file to a new location for processing. This prevents changes to the file while it's being processed. The application writing to the original alert log file should regenerate a new one automatically. After the temporary file has been parsed, you can optionally append the data to a historical log for archival purposes. Using batch mode provides the advantage of knowing what has been completed if there is an interruption to the processing. Throttling messages can also be easier, since you can calculate up front how many alerts are in the queue.

### Example 7-1: Sample Batch Mode Processing

```

# -----
# BATCH MODE PROCESSING
# -----
IDS_LOG="ids.log"           # Alert log file generated
HISTORY_LOG="`date +%Y%m%d`.log" # Archived alerts
TMP_FILE="tmp_idslog.$$ "   # Temporary file during processing

mv ${IDS_LOG} ${TMP_FILE}  # Move the active log somewhere else
touch ${IDS_LOG}           # Recreate an empty log

alert_manager ${TMP_FILE}  # Call a function called 'alert_manager'
                          # and pass it the file to process (in this
                          # case $TMP_FILE)

cat ${TMP_FILE}>>${HISTORY_LOG} # Append the data to today's archive when done

```

Real-time handling instead acts directly on the original log file by opening a pipe that feeds the alerting program. In this case, the alert management program must be designed to accept a continuous flow of data (see 7-3). The advantage, of course, is that you receive alerts as they are generated, rather than waiting until a specified time period has elapsed.

#### Example 7-2: Sample Real-time Processing

```
/bin/nohup tail -lf ids.log | alert_manager &
```

#### Example 7-3: Sample Alert\_Manager Base Code for Real-time Processing

```

#!/bin/sh
while [ 0 = 0 ]; do # While the laws of the known universe exist,
    read ALERT      # read an alert (from example 7-2) into $ALERT
    process_alert   # then process it via a function
done               # called 'process_alert'.

```

The alert manager itself can contain a world of functionality, so I suggest that you provide some command line options to control how alerts are sent. Optionally, you can use a configuration file and base the alert destinations on such things as the target, the type of alert, or the priority.

#### Example 7-4: Sample Command Line Options

```

alert_manager -e john@doe.com -p 555-1234 555-5678 -x fred barney -fg yellow "5
failed logins,su,dino,root" where -e = send email, -p = send page, -x = X-window message,
and spaces delimit multiple destinations

```

You might have noticed at the end of this example, I've added an unusual alert type-"X-window message". Here you can use the HP program *xdialog* or the freely available program *xmessage* to pop up small windows directly on your X11 desktop workstation. These programs also provide `-fg` (foreground) and `-bg` (background) options that allow you to color code your alerts.

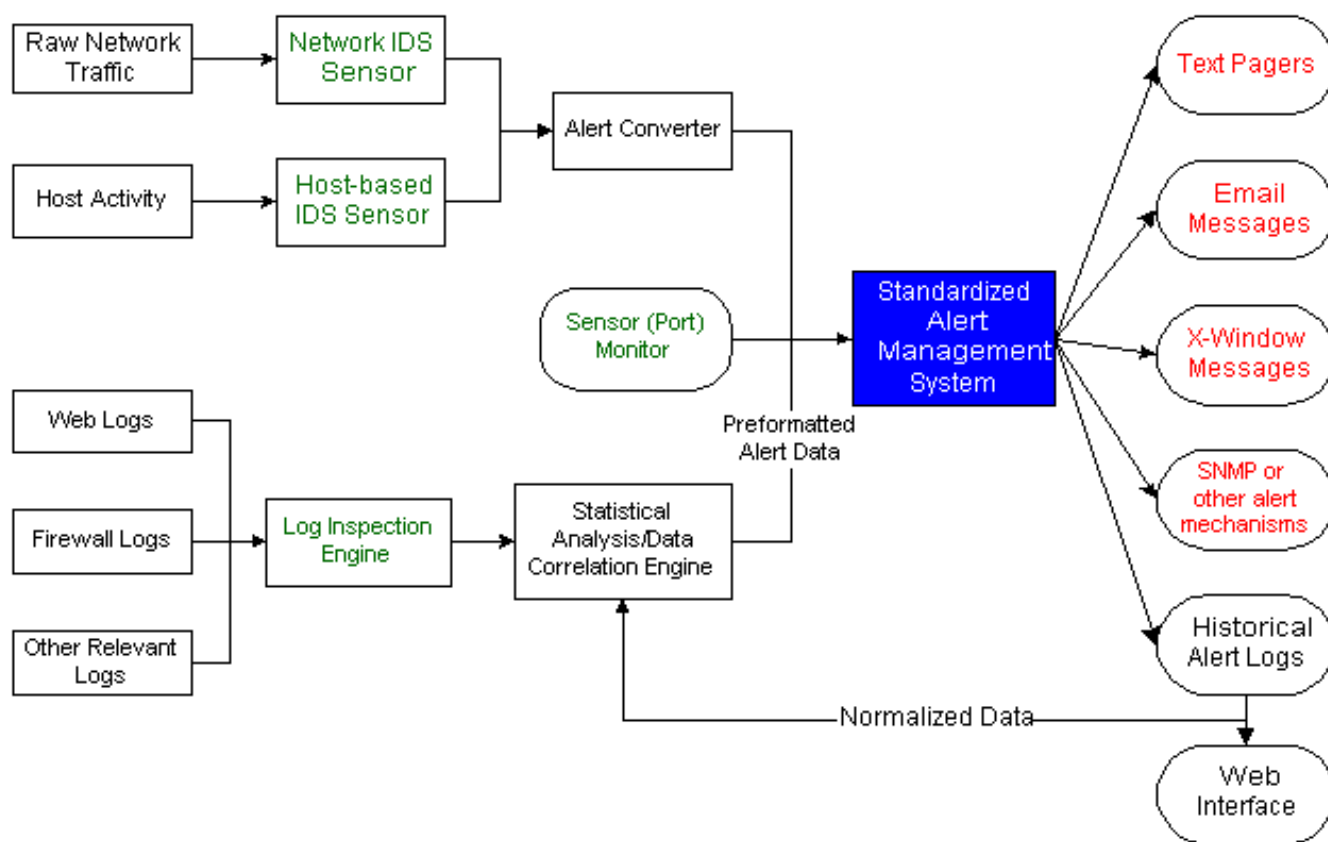
Lastly, write each instance of a notification to a separate log file, so you can maintain a record of when alerts were sent and to what destinations. Be sure to include a portion of the alert message itself, so you can match it with the alert log, if necessary. This allows you to go back and troubleshoot mail or pager issues and helps to determine trends in response and message delivery time.

### Example 7-5: Sample Notification Log Entry

```
Date,Time, Email destination(s), Pager destination(s), X-window host(s), portion of message
03/14/2000,12:21, john@doe.com,555-1234 555-5678,fred barney,5 failed logins
```

The diagram below shows an overview of the entire process, including integration with an independent log inspection engine. Note that the historical alert logs are fed back into a correlation engine to merge data for long-term trend analysis.

### Example 7-6: Integrated Detection-to-Alert Flow



## Automated Resolution and Built-in Investigative Tools

IDS systems typically show source and destination IP addresses for events. Some systems, such as those based on

TCPDump, may also give you a choice of switching between the IP or resolving to hostname (via DNS). In an environment where IP addresses get reassigned frequently, I prefer logging both, in order to know historically who owned an IP at the time of an event. However, I resolve the IP's in a somewhat unique manner by having the alert subsystem use a local cache. If an IP is not cached, the system will check a combination of host files and various DNS's (internal and external) and add it. The cache file itself looks similar to a local host file (IPhostname# Date Time). I add the date and time as a third, commented field to help expire entries. As with alerts, you should build in some throttling to prevent too many name lookups that may be a side effect of Denial of Service attacks.

There are three great advantages to maintaining a cache. First, the alert manager can quickly look up an entry it has seen in the past without needing to hit internal or external DNS's. Second, you can inject entries directly into the cache to give names to hosts that cannot be resolved normally. Third, you can modify hostnames to be more meaningful when they show up in alerts. Just adding this caching technique to our system has cut analysis time drastically. We can now look at an entire day's events and quickly get a good picture of the activity and what 'repeat offenders' are up to.

#### Example 8-1: Normal IDS output

```
10.10.10.10
10.10.22.22
10.55.55.55
10.31.33.7
```

#### Example 8-2: Output using a hosts.cache file after automated name resolution

```
10.10.10.10      Host_unknown*   # 09/28/1999 09:37:47
10.10.22.22      fred            # 09/28/1999 09:40:33
10.55.55.55      dino            # 09/28/1999 10:24:13
10.31.33.7       dialup13.hackem.net # 09/28/1999 11:58:24
```

*\* Note: Placing 'host\_unknown' in the cache prevents wasted resolution time in the future.*

#### Example 8-3: Output using a hosts.cache file after an investigator has updated it

```
10.10.10.10      wilma-Joannes_PC      # 09/28/1999 09:37:47
10.10.22.22      fred-SMTP_Server      # 09/28/1999 09:40:33
10.55.55.55      dino-Vulnerability_Assessment_PC # 09/28/1999 10:24:13
10.31.33.7       dialup13.hackem.net-ScriptKiddie # 09/28/1999 11:58:24
```

#### Example 8-4: Sample code to generate the hosts.cache file:

```

# .....
translate_ip() {          # Translate IPs into Hostnames
# .....
# If the cache file doesn't exist, create it
[ ! -f "$HOST_CACHE" ] && touch $HOST_CACHE

if [ -n "$IP" ]; then
    # Check the cache file first
    HOSTNAME=`egrep "^${IP} " ${HOST_CACHE} | tail -1 | awk '{print $2}'`

    # If not found in the cache, attempt to resolve it
    if [ -z "${HOSTNAME}" ]; then
        # Try the local DNS
        HOSTNAME=`nslookup ${IP} 2>/dev/null|fgrep "Name:"|awk '{print $2}'`
        if [ -z "$HOSTNAME" ]; then
            # If not found in the local DNS, go try an external (Internet) DNS
            LOOKUP=`nslookup <> ${HOST_CACHE}`
        else
            echo "${IP} Host_Unknown # `date '+%Y-%m-%d %T'`" >> ${HOST_CACHE}
        fi
    fi
fi
}

# -----
# MAIN PROGRAM
# -----

HOST_CACHE="/usr/local/hosts.cache"
EXT_DNS="10.10.10.10"          # Enter your nearest external DNS here
IP="$1"                       # Pass the IP at the command line

translate_ip

echo "${IP},${HOSTNAME}"      # Print the result

```

Once you have normalized data being written to your alert logs, you can easily add a web interface to parse the delimited log files and turn them into HTML tables. Herein lies an opportunity to add some investigative tools. First, make your IP addresses HREF anchors, so that clicking them performs a whois. Next, generate a button beside each IP that performs an rDNS scan. If you're feeling daring, you can also add buttons for other functions, such as finger or traceroute. If you cannot do some of these functions directly from your server, visit [SamSpade](#) or [Geektools](#) for online versions.

## Conclusion

So let's assume that you have standardized your system plus built in all the 'extras', and now someone attempts a PHF attack against your web server. The network based IDS picks up the attack (if it doesn't you need a new IDS) and passes it through the alert manager to your text pager. You receive the page in a minute or two and pull up the web interface for more details. The hostname has already been resolved for you (assuming no spoofing) to d2.widgets.com. One click on the rDNS scan button, and you now know that d2 is sandwiched between several web servers, making you suspicious that widgets.com may have been compromised to launch an attack against you. Now, one click on the whois button, and you're on the phone with the sys admin. Total time-5 minutes; total mouse clicks-4.

Okay, so back in the real world, most attacks won't be that simple. But if you find yourself doing these or similar steps every time you get an alert, it makes sense to automate as much of the investigation as possible. Most of your time can probably be better spent performing more in depth analysis, so why not make your intrusion detection system start pulling its weight and doing some of the work for you?

*Robert MacBride currently manages Capital One Financial's Computer Incident Response Team and is leading the development of their integrated intrusion detection systems. His four years in network security engineering ranges from work on authentication systems to host monitoring. Rob also has a degree in Electrical Engineering, as well as eight years of experience in relational database design and UNIX system administration.*

## Relevant Links

[CVE](#)

*Mitre*

[IDWG](#)

*IETF*

[Privacy Statement](#)

Copyright 2006, SecurityFocus